

Laboratório de Linguagens de Programação
Prof. Andrei Rimsa Álvares

Trabalho Prático I

1. Objetivo

O objetivo desse trabalho é desenvolver um interpretador para um subconjunto de uma linguagem de programação conhecida. Para isso foi criada *miniLua*, uma linguagem de programação de brinquedo baseada em Lua (<https://www.lua.org>). Ela possui suporte a tipos dinâmicos lógicos, numéricos (ponto-flutuante), strings e tabelas.

2. Contextualização

A seguir é dado um exemplo de utilização da linguagem *miniLua*. O programa passo a passo uma expressão aritmética dada.

```
-- Tabela com os operadores suportados.
ops = { ["+"] = "add", ["-"] = "sub", ["*"] = "mul", ["/"] = "div" }

-- Tabela de estatísticas com os nomes das operações.
stats = {}
for symbol, op in ops do
    stats[op] = 0
end

-- Uma tabela com as expressões.
exp = {}

--[[
    Entrada de dados de uma expressão, onde cada elemento é dado em uma linha separada.
    O processo termina se a entrada for vazia.
--]]
print("Entre com uma expressão:")
repeat
    v = read(" Proximo elemento: ")
    if v ~= "" then
        exp[#exp+1] = v
    end
until v == ""

-- Cálculo passo a passo da expressão dada.
print("Operações:")
res = exp[1]
for i=2,#exp-1,2 do
    op = ops[exp[i]]
    next = tonumber(exp[i+1]) or 0

    if op then
        if op == "add" then
            tmp = res + next
        elseif op == "sub" then
            tmp = res - next
        elseif op == "mul" then
            tmp = res * next
        elseif op == "div" then
            tmp = res / next
        end
    end
end
```



Laboratório de Linguagens de Programação
Prof. Andrei Rimsa Álvares

```
print(" " .. op .. "(" .. res .. ", " .. next .. "): " .. tmp)
res = tmp

stats[op] = stats[op] + 1
else
  print(" ???(" .. res .. ", " .. next .. "): erro")
  res = 0
end
end

-- Imprimir estatísticas.
print()
print("Estatísticas:")
print(" " .. stats.add .. (stats.add == 1 and " adicao" or " adicoes"))
print(" " .. stats.sub .. (stats.sub == 1 and " substracao" or " subtracoes"))
print(" " .. stats.mul .. (stats.mul == 1 and " multiplicacao" or " multiplicacoes"))
print(" " .. stats.div .. (stats.div == 1 and " divisao" or " divisoes"))
```

calc.ml

A linguagem *miniLua* possui escopo global para as variáveis. Ela suporta os seguintes tipos primitivos: nulo (**nil**), lógico (**false/true**), numérico (números em ponto-flutuante), strings (sequência de caracteres entre aspas duplas) e tabela (estrutura de dados que guarda valores indexados por qualquer tipo, exceto **nil**, separados por vírgulas entre abre e fecha chaves). Valores não inicializados devem ser considerados como **nil**. A linguagem possui conversões implícitas dependendo do operador utilizado (se não for possível fazer a conversão deve-se gerar um erro tempo de execução). Operadores relacionais funcionam apenas com números, exceto os operadores de igualdade (==) e diferença (~=), que funcionam também com outros tipos. Não devem ser feitas conversões implícitas de tipos em expressões condicionais; se os tipos forem diferentes a igualdade obtém falso. Em expressões lógicas, é considerado falso apenas **nil** e a constante lógica **false**.

Tabelas são as estruturas de dados mais importante da linguagem. Ela indexa valores usando índices de qualquer tipo, exceto **nil**. A tabela é dinâmica, ou seja, novos elementos podem ser adicionados a ela e também removidos dela, se a uma chave for atribuída o valor **nil**. Não existem cópias de tabela quando há atribuição de variáveis; variáveis são meros apontadores para elas. Se índices não forem especificados na criação da tabela, assume-se que os valores são indexados a partir do índice 1 (e não em 0 como em outras linguagens). Os valores da tabela podem ser acessados via sintaxe de colchetes (por exemplo, **tbl[1]** ou **tbl["one"]**), ou via sintaxe de propriedades (por exemplo: **tbl.one**). Note que a propriedade tem que ter um nome válido (por exemplo, não é possível usar **tbl.1** ou **tbl.+**).

A linguagem possui comentários de uma linha, onde são ignorados qualquer sequência de caracteres após a sequência -- (menos menos). Ela também possui comentários multilinhas entre --[[e --]]. A linguagem possui as seguintes características:

1) Comandos:

- a. **if**: executar comandos se a expressão for verdadeira.
- b. **while**: repetir comandos enquanto a expressão for verdadeira.
- c. **repeat**: executar comandos e repetir se a expressão for falsa.
- d. **for**: possui duas sintaxes, **numérico** e **genérico**:



Laboratório de Linguagens de Programação

Prof. Andrei Rimsa Álvares

númeroico: *for var=expr1,expr2,expr3 do ... end* (itera de expr1 até expr2 usando expr3 como passo para incrementar var; expr3 é opcional, assume-se o valor 1 se não estiver presente).

genérico: *for var1, var2 in expr do ... end* (expr deve ser uma tabela, onde cada iteração var1 recebe um índice e var2 seu valor correspondente; var2 é opcional, assim apenas a chave var1 é usada).

e. **print:** imprimir na tela com nova linha.

f. **atribuição:** atribuir os valores das expressões do lado direito às expressões do lado esquerdo. As quantidades de ambos os lados não necessariamente precisam ser iguais.

Ex.: *min, max = max, min;* (trocam-se seus valores).

a, b, c = 5, 10; (a recebe 5, b recebe 10 e c recebe nil).

x, y = 1, 2, 3; (x recebe 1, y recebe 2 e o valor 3 não é utilizado).

2) Constantes:

a. **nil:** valor nulo.

b. **Lógico:** valores **false** e **true**.

c. **Número:** valores formados por números em ponto flutuante.

d. **String:** uma sequência de caracteres entre aspas duplas.

e. **Tabela:** elementos separados por vírgulas entre abre e fecha chaves. Os valores são indexados usando sintaxe de colchetes (*tbl["one"]*) ou usando a sintaxe de propriedades (*tbl.one*). O índice pode ser de qualquer tipo, exceto **nil**. A tabela pode ser inicializada vazia, com valores (indexados a partir do índice 1), ou com pares chave/valor, ou com uma combinação dessas.

Ex.: *tbl = {}*

tbl = { "one", "two", "three" }

tbl = { ["one"] = 1, ["two"] = 2, ["three"] = 3 }

3) Valores:

a. Variáveis (começam com **_** ou letras, seguidos de **_** letras ou dígitos).

b. Literais (números, strings e lógicos).

c. Dinâmicos (tabelas).

4) Operadores:

a. **Numéricos:** **+** (adição), **-** (subtração), ***** (multiplicação), **/** (divisão), **%** (resto).

b. **String:** **..** (concatenação)

c. **Lógico:** **==** (igualdade), **~=** (diferença), **<** (menor, entre números), **>** (maior, entre números), **<=** (menor igual, entre números), **>=** (maior igual, entre números), **not** (negação).

d. **Conector:** **and** (E), **or** (OU); ambos usam curto-circuito.

5) Funções:

a. **read:** ler uma linha do teclado, sem o caracter de nova linha (**\n**).

b. **tonumber:** converter número ou string em número, **nil** se não for possível ou for outro tipo.

c. **tostring:** converter lógico, número ou string em string, **nil** se for outro tipo.



Laboratório de Linguagens de Programação
Prof. Andrei Rimsa Álvares

3. Gramática

A gramática da linguagem *miniLua* é dada a seguir no formato de Backus-Naur estendida (EBNF):

```
<code>      ::= { <cmd> }
<cmd>      ::= (<if> | <while> | <repeat> | <for> | <print> | <assign>) [';']

<if>       ::= if <expr> then <code> { elseif <expr> then <code> } [ else <code> ] end

<while>    ::= while <expr> do <code> end
<repeat>   ::= repeat <code> until <expr>
<for>      ::= for <name> (('=' <expr> ',' <expr> [',' <expr>]) | ([',' <name>] in <expr>)) do <code> end

<print>    ::= print '(' [ <expr> ] ')'
<assign>   ::= <lvalue> { ',' <lvalue> } '=' <expr> { ',' <expr> }

<expr>     ::= <rel> { (and | or) <rel> }
<rel>      ::= <concat> [ ('<' | '>' | '<=' | '>=' | '~=' | '==') <concat> ]
<concat>   ::= <arith> { '..' <arith> }
<arith>    ::= <term> { ('+' | '-') <term> }
<term>     ::= <factor> { ('*' | '/' | '%') <factor> }
<factor>   ::= '(' <expr> ')' | [ '-' | '#' | not ] <rvalue>

<lvalue>   ::= <name> { '.' <name> | '[' <expr> ']' }
<rvalue>   ::= <const> | <function> | <table> | <lvalue>

<const>    ::= <number> | <string> | false | true | nil
<function> ::= (read | tonumber | tostring) '(' [ <expr> ] ')

<table>    ::= '{' [ <elem> { ',' <elem> } ] '}'
<elem>     ::= [ '[' <expr> ']' '=' ] <expr>
```

4. Instruções

Deve ser desenvolvido um interpretador em linha de comando que recebe um programa-fonte na linguagem *miniLua* como argumento e executa os comandos especificados pelo programa. Por exemplo, para o programa *calc.ml* deve-se produzir uma saída semelhante a:

```
$ ./mli calc.ml
Usage: ./mrbi [miniLua file]
$ ./mli calc.ml
Entre com uma expressao:
Proximo elemento: 3
Proximo elemento: *
Proximo elemento: 4
Proximo elemento: +
Proximo elemento: 5
Proximo elemento: /
Proximo elemento: 2
```



Laboratório de Linguagens de Programação

Prof. Andrei Rimsa Álvares

Operacoes:
mul(3, 4): 12
add(12, 5): 17
div(17, 2): 8.5

Estatisticas:
1 adicao
0 subtracoes
1 multiplicacao
1 divisao

O programa deverá abortar sua execução, em caso de qualquer erro léxico, sintático ou semântico, indicando uma mensagem de erro. As mensagens são padronizadas indicando o número da linha (2 dígitos) onde ocorreram:

Tipo de Erro	Mensagem
Léxico	Lexema inválido [<i>lexema</i>]
	Fim de arquivo inesperado
Sintático	Lexema não esperado [<i>lexema</i>]
	Fim de arquivo inesperado
Semântico	Operação inválida

Exemplo de mensagem de erro:

```
$ ./mli erro.ml  
03: Lexema não esperado [;]
```

5. Avaliação

O trabalho deve ser feito em grupo de até dois alunos, sendo esse limite superior estrito. O trabalho será avaliado em 15 pontos, onde essa nota será multiplicada por um fator entre 0.0 e 1.0 para compor a nota de cada aluno individualmente. Esse fator poderá estar condicionado a apresentações presenciais a critério do professor. A avaliação é feita exclusivamente executando casos de testes criados pelo professor. Portanto, códigos que não compilam ou não funcionam serão avaliados com nota **ZERO**.

Trabalhos copiados, parcialmente ou integralmente, serão avaliados com nota **ZERO**, sem direito a contestação. Você é responsável pela segurança de seu código, não podendo alegar que outro grupo o utilizou sem o seu consentimento.

6. Submissão

O trabalho deverá ser submetido até as 23:59 do dia 20/12/2021 (segunda-feira) via sistema acadêmico em pasta específica. Não serão aceitos, em hipótese alguma, trabalhos enviados por e-mail ou por quaisquer outras fontes. Para trabalhos feitos em dupla, deve-se criar um arquivo README na raiz do projeto com os nomes dos integrantes da dupla. **A submissão deverá ser feita apenas por um dos integrantes da dupla.**



